Docket No. 15437-0589 *Patent*

UNITED STATES PATENT APPLICATION

FOR

GLOBAL VISIBILITY CONTROLS FOR OPERATING SYSTEM PARTITIONS

INVENTOR:

ANDREW G. TUCKER
JOHN T. BECK
DAVID S. COMAY
ANDREW D. GABRIEL
OZGUR C. LEONARD
DANIEL B. PRICE

PREPARED BY:

HICKMAN PALERMO TRUONG & BECKER LLP
1600 WILLOW STREET
SAN JOSE, CALIFORNIA 95125
(408) 414-1080

"Express Mail" mailing label number EV323351533US

Date of Deposit January 21, 2004

SUN030232NP

GLOBAL VISIBILITY CONTROLS FOR OPERATING SYSTEM PARTITIONS

Inventor(s): ANDREW G. TUCKER, JOHN T. BECK, DAVID S. COMAY, ANDREW D. GABRIEL, OZGUR C. LEONARD, DANIEL B. PRICE

Claim Of Priority

[0001] This application claims benefit of Provisional Application No. 60/469,558, filed May 9, 2003, entitled "OPERATING SYSTEM VIRTUALIZATION," by Andrew G. Tucker, et al., the entire contents of which are incorporated by reference as if fully set forth herein.

Background

[0002] Many of today's computing systems include computing resources that are not fully utilized. Such underutilization provides a potential opportunity to the owners of these systems to obtain greater capacity or cost reduction through improving utilization of these computing resources.

[0003] A number of approaches could be used to address the problem of improving utilization, including consolidation of multiple applications onto a single hardware platform. Consolidation approaches typically attempt to support the co-existence of multiple applications on a single unit of hardware in order to achieve greater function from fewer hardware platforms. A variety of computing resource management techniques could be used for this purpose.

[0004] Such computing resource management extensions, however, must address security and management issues arising from the concurrent execution of multiple applications on a single platform. For example, if web server applications belonging to two or more "untrusting" parties, i.e., market competitors, for example, are co-located on a single hardware platform, neither party will be content with the other party's having access to that party's private information. Some computer system functions, including for example,

facilities to allocate and use hardware resources, i.e., network connections, DASD, output devices, and so forth, file system resources and communications resources could be used by one untrusting party to access the information or applications of another party if access is not controlled. Accordingly, in environments where users do not trust each other to perform system resource related tasks, the system administrator may be burdened with responsibility of performing each action involving critical system resources at significant time and expense.

[0005] One approach to the utilization and security issues arising in consolidation techniques is to partition machine resources among a number of logical partitions (LPARs) or virtual partitions (VPARs), effectively creating multiple machine images on a single platform. Such logical partitioning approaches potentially provide complete isolation among applications based in different machine images. A number of issues arise, however, with logical partitioning approaches. Such approaches may require implementation of hardware support (such as the introduction of an additional privilege level) to isolate privileged programs such as operating system kernels. Also, logical partitioning may require that the system administrator manage the configuration for the logical partitions and the allocation of resources among the logical partitions.

[0006] In another possible approach, one or more instances of operating system images that execute concurrently on a single hardware platform provide a plurality of "Virtual Machines." In such approaches, each virtual machine may be a separate operating system instance that provides isolation for programs running in the virtual machine from other programs running in a second virtual machine. While such virtual machine approaches provide isolation between applications, other issues with such approaches may arise. For example, it may not be necessary, or even desirable to have multiple instances of an entire

operating system for some applications. The complexity of administration and management for different operating systems images may weigh in favor of more simplified approaches.

[0007] Another approach would be to implement compartmentalization into a number of operating system functions. For example, some operating systems employing hierarchical file systems include a function to provide a level of compartmentalization by limiting file system access to a particular process. Such mechanisms, however, also suffer drawbacks. For example, a process's visibility of the file system name space may be limited to a single subtree of the file system in many implementations. Thus, compartmentalization typically does not extend to the process or networking spaces, making observation and interference by other processes possible.

[0008] A yet further approach would be to confine a process and its progeny (i.e., parent and children) to compartmentalized allocations of system resources, i.e., file system, network facilities, and so forth. In this approach, a process placed in such a compartment, termed a "Jail," would have access to allocated system resources, but no visibility nor access to files, processes or network services outside of the Jail. A number of issues arise with the Jails approach, as well. Typically, Jails have no independent existence apart from the process for which the Jail is created. In other words, once the process creating the Jail (and its progeny, if any) terminates, the Jail terminates. Also, a second process cannot "join" a Jail.

Summary

[0010] Some embodiments provide techniques for managing visibility and access to objects by processes in a single kernel instance operating system environment that has been partitioned into a global zone and one or more non-global zones. Visibility and access management includes controlling the ability of processes in one zone to observe objects in another zone and the ability of processes in one zone to access objects in another zone. Isolation and virtualization of processes within a single operating system image may be achieved in one embodiment by providing processes in non-global zones visibility and access exclusively to objects within the non-global zone. In addition to visibility and access to objects in the global zone, processes in the global zone may have permission to view objects in the non-global zones. In some embodiments, processes in the global zone may obtain permission to access objects in the non-global zones.

[0011] The zones comprise persistent virtual environments. Zones are persistent because a zone may exist for longer period of time than the life of any one process in that zone. Zones are virtual environments because each zone provides a semi-autonomous environment to one or more processes that comprises the functional equivalent of a physical machine. In one embodiment, processes in the global zone are allowed to observe processes and other objects in non-global zones, as well as in the global zone. This allows such global zone processes to have system-wide observability. The ability of processes in the global zone to control or send signals to processes in other zones, however, is restricted in one embodiment by use of a new privilege. In one embodiment, processes in the global zone must hold the privilege in order to access or control processes or objects in the non-global zones, as will be described herein.

[0012] In one embodiment, a separate image of a system table of process information is provided to each zone using the file system in order to enable users and processes in the zone to view or access processes or objects in the other zones but to block users from viewing and accessing objects in other zones. In this embodiment, file system permissions can be used to enforce controls on visibility and access of file system related objects. For example, processes executing within a zone may be provided file system access permissions for the file system directories relating to that zone, but will be prevented from accessing other areas of the file system outside of the zone.

[0013] Because many applications assume that a tree like hierarchy of processes exists in the operating system environment, processes in a non-global zone that have parent processes outside of the zone are indicated as having a particular generic parent process, called *init* in one embodiment, that is made visible to processes and users within each zone. Similarly, the scheduler process, called *sched* in one embodiment, is visible to users and processes in each zone. Thus, in one embodiment, process 0 (sched) and process 1 (init) are visible within each zone, including the global zone.

[0014] In one embodiment, the global zone includes a system table in its file system, which includes information about all processes in the system. By accessing this file system, processes and users in the global zone can view the status of the entire operating system environment.

[0015] Before startup, a system administrator configures the zones using administrative commands. In configuring a non-global zone, an administrator may specify a number of different parameters, including, but are not limited to, a zone name, a zone path to the root directory of the zone, specification of one or more file systems to be mounted when the zone is created, specification of zero or more network interfaces, specification of devices to be

configured when the zone is created, and zero or more resource controls to be imposed on the zone.

[0016] At startup, one or more of the configured non-global zones may be booted. When a non-global zone has been configured, it means that an administrator in the global zone has invoked certain services of the kernel to specify all of the configuration parameters for the non-global zone and has saved that configuration in persistent physical storage. During the boot process, portions of file systems are mounted for each individual zone that has been configured in order to instantiate the zones. It is not necessary for the zone to have an active user process running within it in even after it has been booted.

[0017] During runtime, one or more privileges may be checked when access to computing environment objects or computing resources are requested. In addition, the selected mounting of file system resources to portions of the file system associated with individual zones enforces visibility and access limitations on processes in individual zones. In one embodiment, processes running within the non-global zone are limited to viewing and accessing objects only within that non-global zone. In one embodiment, processes associated with the global zone have visibility and access to objects within the global zone, as well as visibility to objects in non-global zones.

[0018] The ability to control or send signals to processes in other zones, however, is restricted by a new privilege, called PRIV_PROC_ZONE in one embodiment. The new privilege allows processes to override the restrictions placed on unprivileged processes. In one embodiment, the restriction that is placed on unprivileged processes in the global zone is that these processes cannot signal or control processes in other zones. This is true even in cases where the user ids of the processes match or the acting process has the PRIV_PROC_OWNER privilege. Also, the PRIV_PROC_ZONE privilege can be removed

from otherwise privileged processes to restrict potentially destructive actions to the global zone.

[0019] Some embodiments can provide secure virtualization and isolation among processes by executing the processes concurrently in one or more discrete zones of the application environment but selectively provide visibility and/or access to data objects and processes among the zones via a global zone. Some embodiments can provide virtualization and isolation among coexisting, concurrently executing processes, without requiring implementation of hardware support (such as the introduction of an additional privilege level) to isolate privileged programs, and without multiple instances of an operating system kernel for some applications.

Brief Description of the Drawings

[0020] Fig. 1 is a functional block diagram of an operating system environment in which one embodiment of the present invention may be implemented.

[0021] Fig. 2A is functional block diagram of example processes in an operating system environment of Fig. 1 in which one embodiment of the present invention may be implemented.

[0022] Fig. 2B is functional block diagram of an example file system for the operating system environment of Fig. 2A in which one embodiment of the present invention may be implemented.

[0023] Fig. 2C is functional block diagram of an example process of obtaining permission to access a non-global zone object in the operating system environment of Fig. 1 in which one embodiment of the present invention may be implemented.

[0024] Figs. 3A – 3H are operational flow diagrams illustrating the operation of one embodiment of the present invention.

[0025] Fig. 4 is an example listing of processes present in one embodiment of the present invention.

[0026] Fig. 5 is a hardware block diagram of an example computer system, which may be used to embody one or more components of an embodiment of the present invention.

Detailed Description of Embodiment(s)

System Overview

[0027] Visibility and access management in some embodiments includes controlling the ability of processes in one zone to observe objects in another zone and the ability of processes in one zone to access objects in another zone. In one embodiment, processes in the global zone are allowed to observe processes and other objects in non-global zones, as well as in the global zone. This allows such global zone processes to have system-wide observability. The ability of processes in the global zone to control or send signals to processes in other zones, however, is restricted in one embodiment by use of a new privilege. In one embodiment, processes in the global zone must hold the privilege in order to access or control processes or objects in the non-global zones, as will be described herein.

[0028] In general, in some embodiments, the present invention provides methods, apparatus and computer program products for managing access to resources in a computer operating system environment. According to one embodiment, a computer based method for managing resources in an operating system environment controlled by a single kernel instance is provided. The method includes creating a global zone and at least one non-global zone. Processes of the global zone may be permitted to view and access objects in the global zone and view objects in non-global zones. Processes of the non-global zone may be permitted to view and access objects only in the non-global zone. The method also includes selectively permitting upon authorized request, a process of the global zone to access objects in a non-global zone.

[0029] In another embodiment, a method for managing resources in a computer operating system includes establishing a global zone and establishing one or more non-global zones. The method further includes selectively limiting visibility and/or access by processes

associated with the global zone to objects within the global zone and select objects within the one or more non-global zones. The method still further includes limiting visibility and access by processes associated with each non-global zone to objects within that non-global zone. The method enables the global zone and the one or more non-global zones to exist concurrently in a single kernel image operating system.

[0030] In one variation, the method includes receiving a request from a requesting process associated with the global zone for visibility and/or access to an object in a non-global zone. The method also includes determining whether the requesting process is authorized for the requested visibility and/or access. If the requesting process is authorized, selectively changing visibility and/or access for the requesting process in accordance with the request is also part of the method. Some embodiments may enable the requesting process to obtain visibility and/or access to objects within the global zone and one or more non-global zones.

[0031] In an alternative variant, a default is provided. For example, in one embodiment, access for processes associated with the global zone defaults to objects within the global zone and visibility for processes associated with the global zone defaults to objects within the global zone and objects within at least one non-global zone. The method includes receiving a request from a requesting process associated with the global zone for access to an object in an non-global zone and, if the requesting process is authorized, selectively changing access of the requesting process in accordance with the request.

[0032] In another embodiment, the method includes receiving an identifier indicating a zone selected from one or more of the global zone and a non-global zone. Mounting file system resources comprising processes to be executed in the zone indicated by the identifier to a portion of a file system associated with the zone indicated by the identifier can also be

part of the method. In some embodiments, the method can enable the processes of the file system resources to obtain visibility and/or access to objects within the zone corresponding to the identifier.

[0033] In one embodiment, file system resources are mounted to a subdirectory of a root directory of a portion of a file system associated with the zone indicated by the identifier so that processes expecting a tree like directory structure to execute within the zone indicated by the identifier.

[0034] In one embodiment, select processes can be made visible to all other processes in the global zone and the non-global zone, enabling processes expecting a tree like calling structure among the processes in the zone to see the specific processes as the root of the tree.

[0035] In a further embodiment, a system is provided. The system comprises a processor and a memory connected with the processor. The memory can be operative to hold at least one of a plurality of program processes, which may include instructions for providing an operating system and instructions for establishing and managing a plurality of zones within a single image of the operating system. Further, the system may include instructions for creating a global zone and one or more non-global zones. Instructions for permitting processes attached to the global zone to view and access objects in the global zone and view objects in non-global zones may also be included in the system. Further, the system may include instructions for permitting processes attached to the non-global zone to view and access objects only in the non-global zone. Instructions for selectively permitting upon authorized request, a process attached to the global zone to access objects in a non-global zone are also included in the system.

[0036] Some embodiments can provide secure virtualization and isolation among processes by executing the processes concurrently in one or more discrete zones of the

application environment but selectively provide visibility and/or access to data objects and processes among the zones via a global zone. Some embodiments can provide virtualization and isolation among coexisting, concurrently executing processes, without requiring implementation of hardware support (such as the introduction of an additional privilege level) to isolate privileged programs, and without multiple instances of an operating system kernel for some applications.

[0037] Fig. 1 illustrates a functional block diagram of an operating system (OS) environment 100 in accordance with one embodiment of the present invention. OS environment 100 may be derived by executing an OS in a general-purpose computer system, such as computer system 500 illustrated in Fig. 5, for example. For illustrative purposes, it will be assumed that the OS is Solaris manufactured by Sun Microsystems, Inc. of Santa Clara, California. However, it should be noted that the concepts taught herein may be applied to any OS, including but not limited to Unix, Linux, Windows™, MacOS™, etc.

[0038] As shown in Fig. 1, OS environment 100 may comprise one or more zones (also referred to herein as partitions), including a global zone 130 and zero or more non-global zones 140. The global zone 130 is the general OS environment that is created when the OS is booted and executed, and serves as the default zone in which processes may be executed if no non-global zones 140 are created. In the global zone 130, administrators and/or processes having the proper rights and privileges can perform generally any task and access any device/resource that is available on the computer system on which the OS is run. Thus, in the global zone 130, an administrator can administer the entire computer system. In one embodiment, it is in the global zone 130 that an administrator executes processes to configure and to manage the non-global zones 140.

[0039] The non-global zones 140 represent separate and distinct partitions of the OS environment 100. One of the purposes of the non-global zones 140 is to provide isolation. In one embodiment, a non-global zone 140 can be used to isolate a number of entities, including but not limited to processes 170, one or more file systems 180, and one or more logical network interfaces 182. Because of this isolation, processes 170 executing in one non-global zone 140 cannot access or affect processes in any other zone. Similarly, processes 170 in a non-global zone 140 cannot access or affect the file system 180 of another zone, nor can they access or affect the network interface 182 of another zone. As a result, the processes 170 in a non-global zone 140 are limited to accessing and affecting the processes and entities in that zone. Isolated in this manner, each non-global zone 140 behaves like a virtual standalone computer. While processes 170 in different non-global zones 140 cannot access or affect each other, it should be noted that they may be able to communicate with each other via a network connection through their respective logical network interfaces 182. This is similar to how processes on separate standalone computers communicate with each other.

[0040] Having non-global zones 140 that are isolated from each other may be desirable in many applications. For example, if a single computer system running a single instance of an OS is to be used to host applications for different competitors (e.g. competing websites), it would be desirable to isolate the data and processes of one competitor from the data and processes of another competitor. That way, it can be ensured that information will not be leaked between the competitors. Partitioning an OS environment 100 into non-global zones 140 and hosting the applications of the competitors in separate non-global zones 140 is one possible way of achieving this isolation.

[0041] In one embodiment, each non-global zone 140 may be administered separately. More specifically, it is possible to assign a zone administrator to a particular non-global zone 140 and grant that zone administrator rights and privileges to manage various aspects of that non-global zone 140. With such rights and privileges, the zone administrator can perform any number of administrative tasks that affect the processes and other entities within that non-global zone 140. However, the zone administrator cannot change or affect anything in any other non-global zone 140 or the global zone 130. Thus, in the above example, each competitor can administer his/her zone, and hence, his/her own set of applications, but cannot change or affect the applications of a competitor. In one embodiment, to prevent a non-global zone 140 from affecting other zones, the entities in a non-global zone 140 are generally not allowed to access or control any of the physical devices of the computer system.

[0042] In contrast to a non-global zone administrator, a global zone administrator with proper rights and privileges may administer all aspects of the OS environment 100 and the computer system as a whole. Thus, a global zone administrator may, for example, access and control physical devices, allocate and control system resources, establish operational parameters, etc. A global zone administrator may also access and control processes and entities within a non-global zone 140.

[0043] In one embodiment, enforcement of the zone boundaries is carried out by the kernel 150. More specifically, it is the kernel 150 that ensures that processes 170 in one non-global zone 140 are not able to access or affect processes 170, file systems180, and network interfaces 182 of another zone (non-global or global). In addition to enforcing the zone boundaries, kernel 150 also provides a number of other services. These services include but are certainly not limited to mapping the network interfaces 182 of the non-global zones 140

to the physical network devices 120 of the computer system, and mapping the file systems 180 of the non-global zones 140 to an overall file system and a physical storage 110 of the computer system. The operation of the kernel 150 will be discussed in greater detail in a later section.

Non-Global Zone States

[0044] In one embodiment, a non-global zone 140 may take on one of four states: (1) Configured; (2) Installed; (3) Ready; and (4) Running. When a non-global zone 140 is in the Configured state, it means that an administrator in the global zone 130 has invoked an operating system utility (in one embodiment, zonecfg(1m)) to specify all of the configuration parameters of a non-global zone 140, and has saved that configuration in persistent physical storage 110. In configuring a non-global zone 140, an administrator may specify a number of different parameters. These parameters may include, but are not limited to, a zone name, a zone path to the root directory of the zone's file system 180, specification of one or more file systems to be mounted when the zone is created, specification of zero or more network interfaces, specification of devices to be configured when the zone is created, and zero or more resource pool associations.

[0045] Once a zone is in the Configured state, a global administrator may invoke another operating system utility (in one embodiment, zoneadm(1m)) to put the zone into the Installed state. When invoked, the operating system utility interacts with the kernel 150 to install all of the necessary files and directories into the zone's root directory, or a subdirectory thereof.

[0046] To put an Installed zone into the Ready state, a global administrator invokes an operating system utility (in one embodiment, zoneadm(1m) again), which causes a zoneadmd process 162 to be started (there is a zoneadmd process associated with each non-global zone).

In one embodiment, zoneadmd 162 runs within the global zone 130 and is responsible for managing its associated non-global zone 140. After zoneadmd 162 is started, it interacts with the kernel 150 to establish the non-global zone 140. In creating a non-global zone 140, a number of operations are performed, including but not limited to assigning a zone ID, starting a zsched process 164 (zsched is a kernel process; however, it runs within the non-global zone 140, and is used to track kernel resources associated with the non-global zone 140), mounting file systems 180, plumbing network interfaces 182, configuring devices, and setting resource controls. These and other operations put the non-global zone 140 into the Ready state to prepare it for normal operation.

[0047] Putting a non-global zone 140 into the Ready state gives rise to a virtual platform on which one or more processes may be executed. This virtual platform provides the infrastructure necessary for enabling one or more processes to be executed within the non-global zone 140 in isolation from processes in other non-global zones 140. The virtual platform also makes it possible to isolate other entities such as file system 180 and network interfaces 182 within the non-global zone 140, so that the zone behaves like a virtual standalone computer. Notice that when a non-global zone 140 is in the Ready state, no user or non-kernel processes are executing inside the zone (recall that zsched is a kernel process, not a user process). Thus, the virtual platform provided by the non-global zone 140 is independent of any processes executing within the zone. Put another way, the zone and hence, the virtual platform, exists even if no user or non-kernel processes are executing within the zone. This means that a non-global zone 140 can remain in existence from the time it is created until either the zone or the OS is terminated. The life of a non-global zone 140 need not be limited to the duration of any user or non-kernel process executing within the zone.

[0048] After a non-global zone 140 is in the Ready state, it can be transitioned into the Running state by executing one or more user processes in the zone. In one embodiment, this is done by having zoneadmd 162 start an init process 172 in its associated zone. Once started, the init process 172 looks in the file system 180 of the non-global zone 140 to determine what applications to run. The init process 172 then executes those applications to give rise to one or more other processes 174. In this manner, an application environment is initiated on the virtual platform of the non-global zone 140. In this application environment, all processes 170 are confined to the non-global zone 140; thus, they cannot access or affect processes, file systems, or network interfaces in other zones. The application environment exists so long as one or more user processes are executing within the non-global zone 140.

[0049] After a non-global zone 140 is in the Running state, its associated zoneadmd 162 can be used to manage it. Zoneadmd 162 can be used to initiate and control a number of zone administrative tasks. These tasks may include, for example, halting and rebooting the non-global zone 140. When a non-global zone 140 is halted, it is brought from the Running state down to the Installed state. In effect, both the application environment and the virtual platform are terminated. When a non-global zone 140 is rebooted, it is brought from the Running state down to the Installed state, and then transitioned from the Installed state through the Ready state to the Running state. In effect, both the application environment and the virtual platform are terminated and restarted. These and many other tasks may be initiated and controlled by zoneadmd 162 to manage a non-global zone 140 on an ongoing basis during regular operation.

Visibility and Access Management

[0050] Visibility and access management includes controlling the ability of processes in one zone to observe objects in another zone and the ability of processes in one zone to access

objects in another zone. In one embodiment, processes in the global zone are allowed to observe processes and other objects in non-global zones, as well as in the global zone. This allows such global zone processes to have system-wide observability. The ability of processes in the global zone to control or send signals to processes in other zones, however, is restricted in one embodiment by use of a new privilege. In one embodiment, processes in the global zone must hold the privilege in order to access or control processes or objects in the non-global zones, as will be described herein.

[0051] Fig. 2A is functional block diagram of example processes in an operating system environment of Fig. 1 in which one embodiment of the present invention may be implemented. As shown in Fig. 2A, during runtime, operating system environment 100 provides a plurality of zones, including non-global zone A 140(a) and non-global zone B 140(b), referred to collectively herein as non-global zones 140. Non-global zones 140 are persistent environments because they may have a lifetime longer than any of the processes associated with them. Further, non-global zones 140 provide a virtualized environment because they are capable of supporting the isolated execution of processes, such as a process A1 174-1(a), executing in non-global zone A 140(a) and a process B1 174-1(b) executing within non-global zone B 140(b). Both of non-global zones 140(a) and 140(b) are able to exist under a single kernel image 150. A process C 232 can execute within global zone 130 of operating system environment 100, as well.

[0052] Further with reference to Fig. 2A, the operation of visibility and access management techniques in one embodiment are illustrated by non-global zone A 140(a), in which process A1 174-1(a) has visibility to and access to an object A 244(a) residing within non-global zone A 140(a) as indicated by a solid line 113(a). Because process A1 174-1(a) is assigned to non-global zone A 140(a), process A1's visibility and access to objects is limited

to only those objects assigned to non-global zone A 140(a). Thus, process A1 174-1(a) is prohibited from viewing or accessing objects C 234 of global zone 130, as indicated by a small dotted line 115(a). Similarly, process B1 174-1(b), assigned to non-global zone B 140(b), is limited to viewing and accessing only those objects assigned to non-global zone B 140(b), such as object B 244(b), for example. Further, process A1 and process B1 are prohibited from viewing or accessing objects in each other's non-global zone. Global zone process C 232 is provided visibility and access to object C 234 in global zone 130, as indicated by solid line 117. In one embodiment, since process C 232 is assigned to global zone 130, process C 232 is able to view objects in the non-global zones, such as object A 244(a) and object B 244(b) as indicated by alternating dashed and dotted line 119. In one embodiment, process C 232 is permitted to access object A 244 or object B 244 provided that process C232 obtains further privileges as will be described herein below.

[0053] Fig. 2B is functional block diagram of an example file system for the operating system environment of Fig. 2A in which one embodiment of the present invention may be implemented. As shown in Fig. 2B, file system 180 of Fig. 1 comprises a number of directories arranged in a hierarchical tree like structure. For example, in non-global zone A 140(a), file system 180(a) is mounted at a zone root directory 290(a). In operating system nomenclature, the root directory is signified by a slash ("/"). Because root directory 290(a) is a zone root directory, it will appear to processes within the non-global zone A 140(a) to be the root directory. Directory 290(a) is a subdirectory of an /AUX0 directory 291, which is part of the file system 180 accessible by processes in the global zone 130. From the point of view of a process in the global zone 130, the directory 290(a) is directory /AUX0/zone A 290(a).

[0054] In one embodiment, the zone's root directory is distinct from the directory set by *chroot*, a system command for establishing a root directory, for the processes within the zone. Both restrictions are checked in one embodiment, when a process is traversing pathname components. This enables *chroot* to be used within a zone, but if a process escapes from its *chroot* restriction, that process will still be unable to escape the zone restriction.

[0055] Zone root directory 290(a) comprises one or more subdirectories, such as/Proc 292(a), /USR 293(a), and /ETC 294(a). This is not an exhaustive list and other subdirectories may also be included in root directory 290(a). These subdirectories may further have subdirectories themselves. For example, the directory /USR/SBIN 295(a) is a subdirectory of the directory /USR 293(a). A number of processes running in non-global zone A such as a process A1 174-1(a), process A2 174-2(a) and process A3 174-3(a) are instantiated from files stored in directory /USR/SBIN 295(a), as is indicated by solid lines 104(a), 106(a) and 108(a).

[0056] Referring now to non-global zone B 140(b) of Fig. 2B, a separate instance of a file system 180(b) is mounted at root directory 290(b). Root directory 290(b) also comprises one or more subdirectories storing data and/or files of processes such as process B1 174-1(b), process B2 174-2(b) and process B3 174-3(b). In the embodiment of Fig. 2B, the directories and associated structure of the file system 180(b) of non-global zone B 140(b) is closely analogous to the portions of file system 180(a) of non-global zone A 140(a), however, this is not necessarily always the case, and implementation specific alternatives will exist. For example, in some embodiments, file system 180(a) and file system 180(b) may be mapped to a single physical storage in what is known as a loop back file system in one embodiment.

[0057] Global zone 130 includes a file system root directory 290 of the operating system environment 100. The file system of global zone 130 also includes one or more

subdirectories, such as for example as /PROC 292, /USR 293, and /ETC 294. In one embodiment, root directory / 290 includes a sub-directory /AUX0 291, which has as its subdirectories directory /AUX0/zone A 290(a) and directory /AUX0/zone B 290(b) to implement the zone root directories /290(a) and / 290(b).

[0058] The process file system, /PROC or *procfs*, is a special purpose file system that implements a view of a process table, normally resident in the memory of the kernel 150 of operating system environment 100, inside the file system 180 in the /PROC directory. The /PROC file system is a virtual file system; it is not associated with a block format storage device, but rather exists in memory. The files in the /PROC file system enable processes executing under control of the operating system environment 100 to access information about other active processes. The information in the /PROC file system is normally accessed by programs that provide information about processes running under the operating system environment 100, such as ps(1) utility, for example. In one embodiment, the ps utility displays a header line followed by lines containing information about processes that have controlling terminals, which is sorted by controlling terminal then by process ID.

[0059] In one embodiment, the /proc file system is enhanced to provide process visibility and access control based upon the zone association of processes running under operating system environment 100 In one embodiment, process access control can be provided based on the location of the mount; a /proc file system mounted within the part of the file system hierarchy rooted at the root directory of a zone in the Ready or Running state will include only processes within that zone. Referring again to Fig. 2B, zone root directory / 290(a) includes a subdirectory /PROC 292(a), which comprises the /PROC file system for zone A. Analogously, zone B includes a subdirectory /PROC 292(b) mounted to the zone root directory 290(b) for zone B.

[0060] In one embodiment, file system 180 is mounted to /PROC 292(a) of zone root directory 290(a) so that processes running in non-global zone A such as process A1 174-1(a), process A2 174-2(a) and process A3 174-3(a), are able to access information in /PROC 292(a) relating exclusively to the processes associated with zone A. This relationship is indicated in Fig. 2B by dotted lines 103(a) and 105(a) (a dotted line for process A3 174-3 has been omitted from Fig. 2B for clarity). Analogously, zone B includes a subdirectory /PROC 292(b) mounted to the zone root directory 290(b) for zone B. Zone B's /PROC 292(b) is accessible to processes running in non-global zone B such as process B1 174-1(b), process B2 174-2(b) and process B3 174-3(b). These processes will be restricted to viewing information relating exclusively to processes within zone B by the /PROC 292(b) image of the process table.

[0061] In one embodiment, mounts issued from within a non-global zone will only display processes within that zone, since these processes must be mounted within the part of the file system hierarchy accessible from that non-global zone. Mounts issued from the global zone, which are not within the part of the file system hierarchy belonging to any ready or running zone, will reflect all processes in the system.

[0062] Because many applications assume that a tree like hierarchy of processes exists in the operating system environment, processes in a non-global zone that have parent processes outside of the zone are indicated as having a parent process, called *zsched* in one embodiment, which is a kernel process created when the zone is placed in the Ready state. The *zsched* process appears to processes in the non-global zones to be its own parent in one embodiment. Thus, in one embodiment, process 0 (*sched*) and process 1 (*init*) are not visible within the non-global zones.

[0063] In one embodiment, the files exported by *procfs* are enhanced to include data about the zone with which each process is associated. In particular, a zone id may be added to data structures containing process information (available by reading the corresponding files in *procfs*). In one embodiment, these new additions enable processes in the global zone to determine the zone associations of processes they are observing or controlling.

[0064] As discussed above with reference to Fig. 2B, processes in the global zone are permitted to observe processes and other objects in non-global zones in one embodiment. This allows such processes to have system-wide observability. The ability to control or send signals to processes in other zones, however, is restricted by a new privilege, which must be obtained by a process in the global zone in order to access or control an object in a non-global zone in one embodiment.

[0065] Fig. 2C is functional block diagram of an example process of obtaining permission to access a non-global zone object in the operating system environment of Fig. 1 in which one embodiment of the present invention may be implemented. As shown by fig. 2C, process C 232 performs processing to obtain appropriate privilege from kernel 150 in order to access object A 244(a). The dotted line 123 indicates a request by process C 232 for an additional privilege from kernel 150 in order to be able to access object A 244(a). A dotted line 125 indicates a granting by kernel 150 of the appropriate privilege to process C 232. In one embodiment, the new privilege allows processes to override the restrictions placed on unprivileged processes. (In this case, the restriction is that unprivileged processes in the global zone cannot signal or control processes in other zones.) Further, in one embodiment, the privilege can be removed from a privileged process in order to restrict possibly destructive actions.

[0066] In an alternative embodiment, filtering based on the zone context of the opening process, rather than through use of a mount as described above with reference to Fig. 2B, can be used to limit access to certain processes within an instance of *procfs*. When a process in the global zone opens a *procfs* instance associated with another zone, it would actually see all processes in the system rather than just the ones associated with that zone. This embodiment differs from the embodiment discussed above with reference to Fig. 2B, in which a given *procfs* instance will export the same processes regardless of the context of the reader.

Sample Operation

[0067] A sample operation of the operating system 100 in accordance with one embodiment of the present invention will now be described. In the following discussion, reference will be made to the system diagram of Figs. 2A - 2C and the flow diagrams of Figs. 3A – 3H.

[0068] Fig. 3A is an operational flow diagram illustrating the operation of one embodiment of the present invention. In the embodiment discussed with reference to Fig. 3A, in block 302 global zone 130 is established. In block 304 non-global zone 140 is established. In block 310, visibility and/or access by process 232 associated with the global zone 130 is limited to object 234 associated with global zone 130, and selectively limited to object 244 in non-global zone 140, as is discussed in further detail with reference to Fig. 3B. In block 320, visibility and/or access by process 174-1 associated with non-global zone 140 is limited to object 244 in non-global zone 140, as is discussed in further detail with reference to Fig. 3C.

[0069] Referring to Fig. 3B, which is an operational flow diagram illustrating the operation of block 310 of Fig. 3A in one embodiment of the present invention, in block 312,

visibility of object 234, in global zone 130, by a process 232 in the global zone 130 is enabled. Further, in that same block 312, visibility of object 244 in the non-global zone 140 by process 232 in the global zone 130 is enabled, as is discussed in further detail with reference to Fig. 3D. In block 314, access to object 234 in a global zone 130 a process 232 in the global zone 130 is enabled. Access to object 244 by process 232 in the global zone 130, however, is selectively restricted, as is discussed in further detail with reference to Fig. 3E.

[0070] Now referring to Fig. 3C, which is an operational flow diagram illustrating the operation of block 320 one Fig. 3A in one embodiment of the present invention, in block 322 visibility of object 244 in non-global zone 140 by process 174-1 in the non-global zone 140 is enabled. Visibility of the object 234 in the global zone 130 by process 174-1 in the non-global zone 140, however, is restricted, as is discussed in further detail with reference to Fig. 3F. In block 324, access to object 244 in non-global zone 140 by a process 174-1 in the non-global zone 140 is enabled. In that same block 324, access to object 234 in the global zone 130 by a process 174-1 in the non-global zone 140 is restricted, as is discussed in further detail with reference to Fig. 3G. In one embodiment, processing such as that described with reference to Fig. 3C blocks a process inside a zone, other than the global zone, from entering any other zones. This prevents a process running with privilege or authority (such as "super-user") in a zone from escaping the zone's root directory restriction in a manner similar to what is possible in conventional systems using the *chroot* command. In one embodiment, the zone's root directory is distinct from the directory set by *chroot* for processes within the zone. In one embodiment, both restrictions are checked in one embodiment, when a process is traversing pathname components. This enables *chroot* to be used within a zone, but if a process escapes from its *chroot* restriction, that process will still be unable to escape the zone restriction.

[0071] Referring to Fig. 3D, which is an operational flow diagram illustrating the operation of block 312 of Fig. 3B in one embodiment of the present invention, in block 332 an identifier associated with global zone 130 is received. Then in block 334, the process table from an operating system kernel 150 is reflected to a subdirectory of the root directory of the file system associated with the operating system.

[0072] Referring to Fig. 3E, which is an operational flow diagram illustrating the operation of block 314 of Fig. 3B in one embodiment of the present invention, in block 342 a request from a process 232 associated with global zone 130 to access an object is received. In block 344, the step of determining based upon a zone identifier, whether the request from process 232 associated with global zone 130 is attempting to access an object associated with a zone other than global zone 130. In block 346, a test is performed to see if the zone ID of the requesting process and the zone ID of the object to be accessed match. If the zone IDs match, then in block 348, the request from process 232 associated with global zone 130 to access an object 234 is permitted. Otherwise, if the zone IDs do not match, then in block 350 a test is performed to see if the requesting process 232 is privileged. If the requesting process is privileged, then in block 352 the request from process 232 associated with global zone 130 to access an object 244(a) or object 244(b) of non-global zone 140(a) or zone 140(b) is permitted. Otherwise if the process 232 is not privileged, in block 354, the request from the process 232 is denied. In one embodiment, process 232 may request additional privilege using a process illustrated Fig. 3H.

[0073] Referring to Fig. 3F, which is an operational flow diagram illustrating the operation of block 322 of Fig. 3C in one embodiment of the present invention, in block 362, identifier associated with non-global zone 140 is received. In block 364, a portion of a process table from an operating system kernel selected based upon the identifier is reflected

to a subdirectory of a root directory of a portion of a file system associated with the non-global zone 140. Then in block 366, a process assigned to the non-global zone 140 is limited to having access to the portion of the file system associated with the non-global zone 140.

[0074] Referring to Fig. 3G, which is an operational flow diagram illustrating the operation of block 324 of Fig. 3C in one embodiment of the present invention, in block 372 a request from a process 174-1(a) associated with non-global zone 140(a) to access an object is received. In block 374, it is determined based upon the zone identifier whether the request from process 174-1(a) associated with non-global zone 140(a) is attempting to access an object associated with the zone other than not global zone 140(a). In block 376, a test of whether the zone ID of the process making the request and the zone ID of the object to be accessed match. If the zone ID's match then in block 378, the request from the process 174-1(a) is permitted. Otherwise, if the zone IDs do not match, then in block 380 the request from process 174-1(a) is denied.

[0075] In the event that a request by a process 232 in the global zone 130 to access an object 244 in the global zone 140 has been denied, as was illustrated by block 354 of Fig. 3E, a request privilege process may be executed. As shown by Fig. 3H, which is an operational flow diagram illustrating the operation of one embodiment of the present invention, in block 392, a request by the process 232 assigned to global zone 130 for permission to access object 244 associated with non-global zone 140 is received. In block 394, a PRIV_PROC_ZONE privilege is associated with the process 232 assigned to global zone 130. The process 232 now has the capability to obtain access to the object 244 in the non-global zone 140 in accordance with processing illustrated by reference to Fig. 3E in one embodiment.

[0076] Fig. 4 is an example listing of processes present in one embodiment of the present invention. As shown in Fig. 4, shell commands may be used to retrieve information about

the zones in existence and processes within the zones in one embodiment. A command 402 is a *zoneadm(1M)* command to observe zones existing in the computer system 100. The *zoneadm* command provides one way to manage zones that have been configured in computer system 100. In command line 402, the *zoneadm* command is invoked using a *global#* prefix, a *list* parameter and a *–v* flag. The *global#* prefix indicates that the 402 is directed to the global zone. The *list* parameter and the *–v* flag indicate that the command is requesting a listing of processes and that the list be "verbose," respectively. Fig. 4 illustrates a response 404 to the *zoneadm* command 402 which indicates that two zones are presently established in computer system 100: a first zone, named *global*, and a second zone, named *my-zone*, which is a non-global zone. Both zones are in the Running state and the path of the root directory of the global zone is /, while the path of the root directory of *my-zone*, is /aux0/my-zone.

[0077] The next two commands and responses illustrate the visibility of process 0 *sched* and 1 *init* which are made visible in the global zone exclusively in one embodiment. A second command 406, which is a *ps* command addressed to the global zone 140, instructs the zone to return the identities of processes visible or known to the global zone 140.

[0078] The ps utility displays a header line followed by lines containing information about the processes in the system 100. This information is sorted by process ID. The information displayed is selected based on a set of keywords (such as -L -O and -o options).

[0079] The global zone 140 responds with a response 408 to the *ps* command 406 with a series of lines that indicate a process id (PID) in a first column, a zone id (ZONEID) and a name of the process, including the directory path (COMMAND) for each of the processes known to the zone 140. A third command 410, also a *ps* command, is appended to a *zlogin* command directed to the non-global zone 140, *my-zone*, to instruct the non-global zone 140

to return the identities of processes visible or known to the non-global zone 140. The non-global zone 140 responds with a response 412 to the *ps* command 410 with a series of lines indicating a process id, a zone id and a name of the process, including its directory path, for each of the processes known to the zone 140. A comparison of response 408 and response 412 from the global and non-global zones, respectively, indicates that the processes 0 *sched* and 1 *init* are made visible in the global zone exclusively, in the illustrated embodiment. In one embodiment, any process in a non-global zone having a parent that is outside of the non-global zone containing the process appears to be parented by process *zsched.* The *zsched* process will appear to be parented by itself in one embodiment. Accordingly, legacy applications that expect a tree structure of processes, i.e., all processes would have a parent-child relationship with a common origin, can continue to function without modification even when executed in a zone in which unrelated processes coexist. Further, such embodiments do not need to expose the process id of the real parent process, which may belong to a global zone, for example. In this way, process information can be isolated to process information of only processes within a non-global zone, eliminating potential exposure of zone-external information to processes in the non-global zone.

Hardware Overview

[0080] Fig. 5 is a block diagram that illustrates a computer system 500 upon which an embodiment of the invention may be implemented. Computer system 500 includes a bus 502 for facilitating information exchange, and one or more processors 504 coupled with bus 502 for processing information. Computer system 500 also includes a main memory 506, such as a random access memory (RAM) or other dynamic storage device, coupled to bus 502 for storing information and instructions to be executed by processor 504. Main memory 506 also may be used for storing temporary variables or other intermediate information during

execution of instructions by processor 504. Computer system 500 may further include a read only memory (ROM) 508 or other static storage device coupled to bus 502 for storing static information and instructions for processor 504. A storage device 510, such as a magnetic disk or optical disk, is provided and coupled to bus 502 for storing information and instructions.

[0081] Computer system 500 may be coupled via bus 502 to a display 512, such as a cathode ray tube (CRT), for displaying information to a computer user. An input device 514, including alphanumeric and other keys, is coupled to bus 502 for communicating information and command selections to processor 504. Another type of user input device is cursor control 516, such as a mouse, a trackball, or cursor direction keys for communicating direction information and command selections to processor 504 and for controlling cursor movement on display 512. This input device typically has two degrees of freedom in two axes, a first axis (e.g., x) and a second axis (e.g., y), that allows the device to specify positions in a plane.

[0082] In computer system 500, bus 502 may be any mechanism and/or medium that enables information, signals, data, etc., to be exchanged between the various components. For example, bus 502 may be a set of conductors that carries electrical signals. Bus 502 may also be a wireless medium (e.g. air) that carries wireless signals between one or more of the components. Bus 502 may also be a medium (e.g. air) that enables signals to be capacitively exchanged between one or more of the components. Bus 502 may further be a network connection that connects one or more of the components. Overall, any mechanism and/or medium that enables information, signals, data, etc., to be exchanged between the various components may be used as bus 502.

[0083] Bus 502 may also be a combination of these mechanisms/media. For example, processor 504 may communicate with storage device 510 wirelessly. In such a case, the bus 502, from the standpoint of processor 504 and storage device 510, would be a wireless medium, such as air. Further, processor 504 may communicate with ROM 508 capacitively. In this instance, the bus 502 would be the medium (such as air) that enables this capacitive communication to take place. Further, processor 504 may communicate with main memory 506 via a network connection. In this case, the bus 502 would be the network connection. Further, processor 504 may communicate with display 512 via a set of conductors. In this instance, the bus 502 would be the set of conductors. Thus, depending upon how the various components communicate with each other, bus 502 may take on different forms. Bus 502, as shown in Fig. 5, functionally represents all of the mechanisms and/or media that enable information, signals, data, etc., to be exchanged between the various components.

[0084] The invention is related to the use of computer system 500 for implementing the techniques described herein. According to one embodiment of the invention, those techniques are performed by computer system 500 in response to processor 504 executing one or more sequences of one or more instructions contained in main memory 506. Such instructions may be read into main memory 506 from another machine-readable medium, such as storage device 510. Execution of the sequences of instructions contained in main memory 506 causes processor 504 to perform the process steps described herein. In alternative embodiments, hard-wired circuitry may be used in place of or in combination with software instructions to implement the invention. Thus, embodiments of the invention are not limited to any specific combination of hardware circuitry and software.

[0085] The term "machine-readable medium" as used herein refers to any medium that participates in providing data that causes a machine to operation in a specific fashion. In an

embodiment implemented using computer system 500, various machine-readable media are involved, for example, in providing instructions to processor 504 for execution. Such a medium may take many forms, including but not limited to, non-volatile media, volatile media, and transmission media. Non-volatile media includes, for example, optical or magnetic disks, such as storage device 510. Volatile media includes dynamic memory, such as main memory 506. Transmission media includes coaxial cables, copper wire and fiber optics, including the wires that comprise bus 502. Transmission media can also take the form of acoustic or light waves, such as those generated during radio-wave and infra-red data communications.

[0086] Common forms of machine-readable media include, for example, a floppy disk, a flexible disk, hard disk, magnetic tape, or any other magnetic medium, a CD-ROM, any other optical medium, punchcards, papertape, any other physical medium with patterns of holes, a RAM, a PROM, and EPROM, a FLASH-EPROM, any other memory chip or cartridge, a carrier wave as described hereinafter, or any other medium from which a computer can read.

[0087] Various forms of machine-readable media may be involved in carrying one or more sequences of one or more instructions to processor 504 for execution. For example, the instructions may initially be carried on a magnetic disk of a remote computer. The remote computer can load the instructions into its dynamic memory and send the instructions over a telephone line using a modem. A modem local to computer system 500 can receive the data on the telephone line and use an infra-red transmitter to convert the data to an infra-red signal. An infra-red detector can receive the data carried in the infra-red signal and appropriate circuitry can place the data on bus 502. Bus 502 carries the data to main memory 506, from which processor 504 retrieves and executes the instructions. The instructions

received by main memory 506 may optionally be stored on storage device 510 either before or after execution by processor 504.

[0088] Computer system 500 also includes a communication interface 518 coupled to bus 502. Communication interface 518 provides a two-way data communication coupling to a network link 520 that is connected to a local network 522. For example, communication interface 518 may be an integrated services digital network (ISDN) card or a modem to provide a data communication connection to a corresponding type of telephone line. As another example, communication interface 518 may be a local area network (LAN) card to provide a data communication connection to a compatible LAN. Wireless links may also be implemented. In any such implementation, communication interface 518 sends and receives electrical, electromagnetic or optical signals that carry digital data streams representing various types of information.

[0089] Network link 520 typically provides data communication through one or more networks to other data devices. For example, network link 520 may provide a connection through local network 522 to a host computer 524 or to data equipment operated by an Internet Service Provider (ISP) 526. ISP 526 in turn provides data communication services through the world wide packet data communication network now commonly referred to as the "Internet" 528. Local network 522 and Internet 528 both use electrical, electromagnetic or optical signals that carry digital data streams. The signals through the various networks and the signals on network link 520 and through communication interface 518, which carry the digital data to and from computer system 500, are exemplary forms of carrier waves transporting the information.

[0090] Computer system 500 can send messages and receive data, including program code, through the network(s), network link 520 and communication interface 518. In the

Internet example, a server 530 might transmit a requested code for an application program through Internet 528, ISP 526, local network 522 and communication interface 518.

[0091] The received code may be executed by processor 504 as it is received, and/or stored in storage device 510, or other non-volatile storage for later execution. In this manner, computer system 500 may obtain application code in the form of a carrier wave.

[0092] In the foregoing specification, it should be noted that although the invention has been described with reference to one embodiment, it should not be construed to be so limited. Various modifications may be made by those of ordinary skill in the art with the benefit of this disclosure without departing from the spirit of the invention. Thus, the invention should not be limited by the embodiments used to illustrate it but only by the scope of the issued claims. The specification and drawings are, accordingly, to be regarded as illustrative rather than limiting.
